Carnegie Mellon University

Parallel Execution

Lecture #14

Database Systems
15-445/15-645
Fall 2018

AP
Andy Pavlo
Computer Science
Carnegie Mellon Univ.

ADMINISTRIVIA

Project #3 is due Monday October 19th

Project #4 is due Monday December 10th

Homework #4 is due Monday November 12th

UPCOMING DATABASE EVENTS

BlazingDB Tech Talk

→ Thursday October 25th @ 12pm

→ CIC - 4th floor (ISTC Panther Hollow Room)

Brytlyt Tech Talk

→ Thursday November 1st @ 12pm

→ CIC - 4th floor (ISTC Panther Hollow Room)

WHY CARE ABOUT PARALLEL EXECUTION?

Increased performance.
→ Throughput
→ Latency

Increased availability.

Potentially lower TCO.

PARALLEL VS. DISTRIBUTED

Database is spread out across multiple resources to improve parallelism.

Appears as a single database instance to the application.
→ SQL query for a single-node DBMS should generate same result on a parallel or distributed DBMS.

PARALLEL VS. DISTRIBUTED

Parallel DBMSs:

→ Nodes are physically close to each other.
→ Nodes connected with high-speed LAN.
→ Communication cost is assumed to be small.

Distributed DBMSs:

→ Nodes can be far from each other.
→ Nodes connected using public network.
→ Communication cost and problems cannot be ignored.

INTER- VS. INTRA-QUERY PARALLELISM

Inter-Query: Different queries are executed concurrently.
→ Increases throughput & reduces latency.

Intra-Query: Execute the operations of a single query in parallel.
→ Decreases latency for long-running queries.

TODAY'S AGENDA

Process Models

Execution Parallelism

I/O Parallelism

PROCESS MODEL

A DBMS's **process model** defines how the system is architected to support concurrent requests from a multi-user application.

A **worker** is the DBMS component that is responsible for executing tasks on behalf of the client and returning the results.

PROCESS MODELS

Approach #1: Process per DBMS Worker

Approach #2: Process Pool

Approach #3: Thread per DBMS Worker

PROCESS PER WORKER

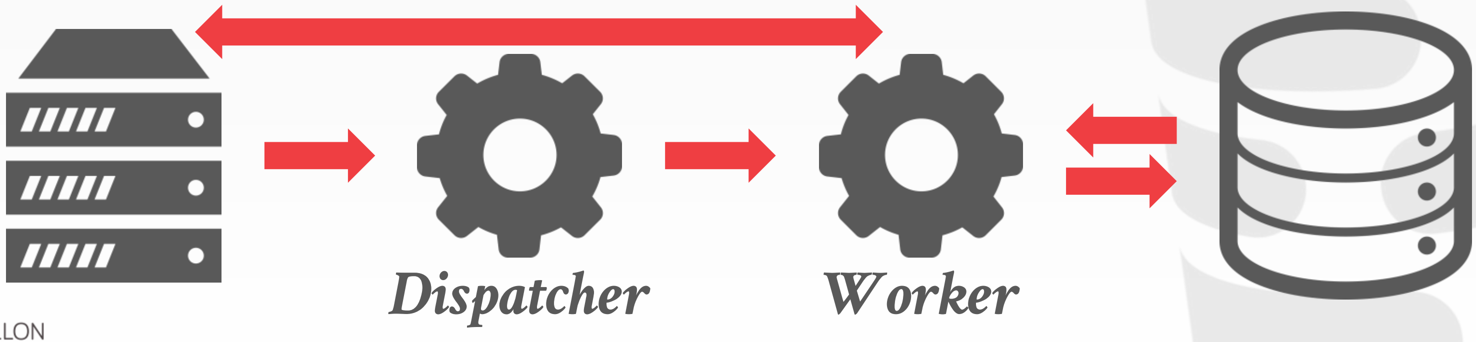
Each worker is a separate OS process.

→ Relies on OS scheduler.

→ Use shared-memory for global data structures.

→ A process crash doesn't take down entire system.

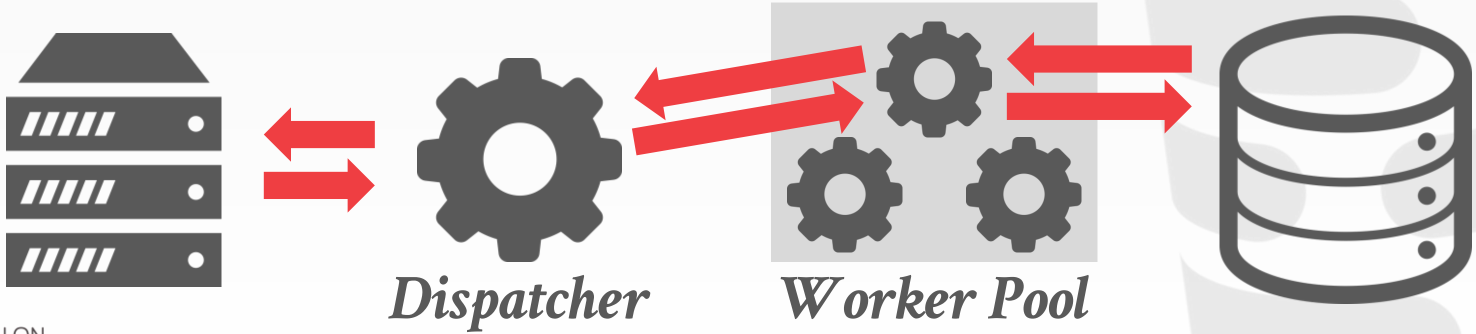
→ Examples: IBM DB2, Postgres, Oracle



PROCESS POOL

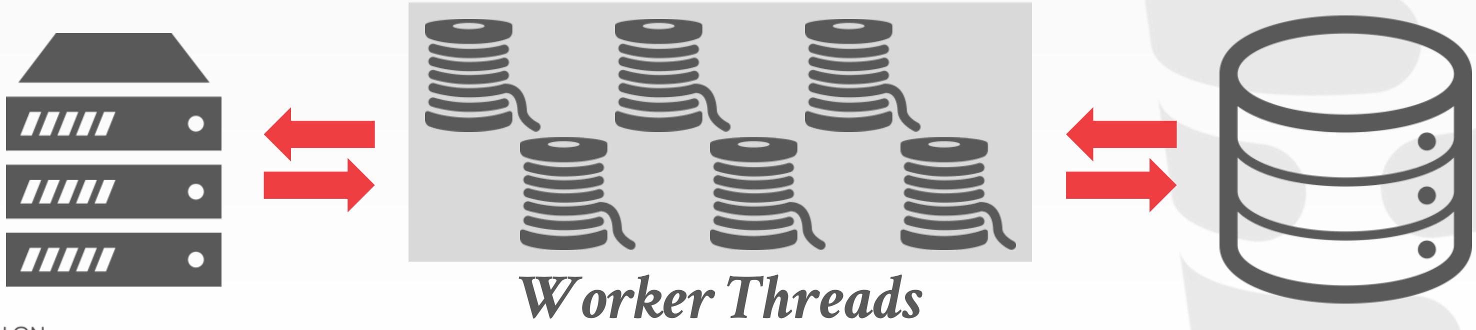
A worker uses any process that is free in a pool

→ Still relies on OS scheduler and shared memory.
→ Bad for CPU cache locality.
→ Examples: IBM DB2, Postgres (2015)



THREAD PER WORKER

Single process with multiple worker threads.
→ DBMS has to manage its own scheduling.
→ May or may not use a dispatcher thread.
→ Thread crash (may) kill the entire system.
→ Examples: IBM DB2, MSSQL, MySQL, Oracle (2014)



PROCESS MODELS

Using a multi-threaded architecture has several advantages:

→ Less overhead per context switch.
→ Don't have to manage shared memory.

The thread per worker model does not mean that you have intra-query parallelism.

I am not aware of any new DBMS built in the last 10 years that doesn't use threads.

SCHEDULING

For each query plan, the DBMS has to decide where, when, and how to execute it.

→ How many tasks should it use?
→ How many CPU cores should it use?
→ What CPU core should the tasks execute on?
→ Where should a task store its output?

The DBMS ***always*** knows more than the OS.

INTER-QUERY PARALLELISM

Improve overall performance by allowing multiple queries to execute simultaneously.

If queries are read-only, then this requires little coordination between queries.

If queries are updating the database at the same time, then this is hard to do this correctly.
→ Need to provide the illusion of isolation.
→ We will discuss more next week.

INTRA-QUERY PARALLELISM

Improve the performance of a single query by executing its operators in parallel.

→ **Approach #1: Intra-Operator**

→ **Approach #2: Inter-Operator**

These techniques are not mutually exclusive.

There are parallel algorithms for every relational operator.

INTRA-OPERATOR PARALLELISM

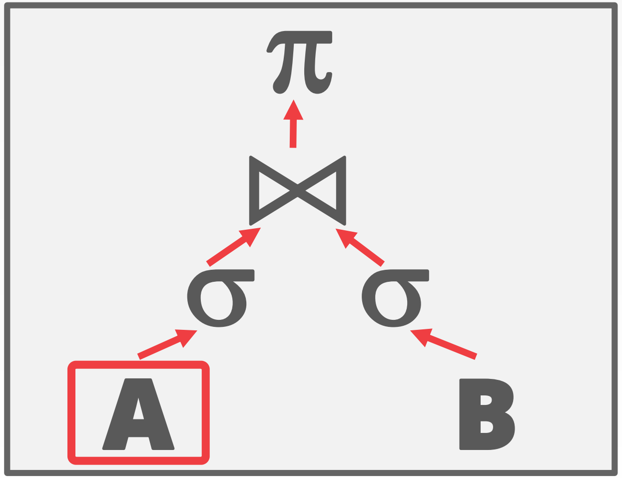
Approach #1: Intra-Operator (Horizontal)

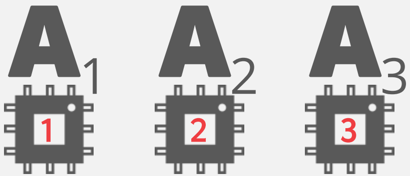
→ Operators are decomposed into independent instances that perform the same function on different subsets of data.

The DBMS inserts an **exchange** operator into the query plan to coalesce results from children operators.

INTRA-OPERATOR PARALLELISM

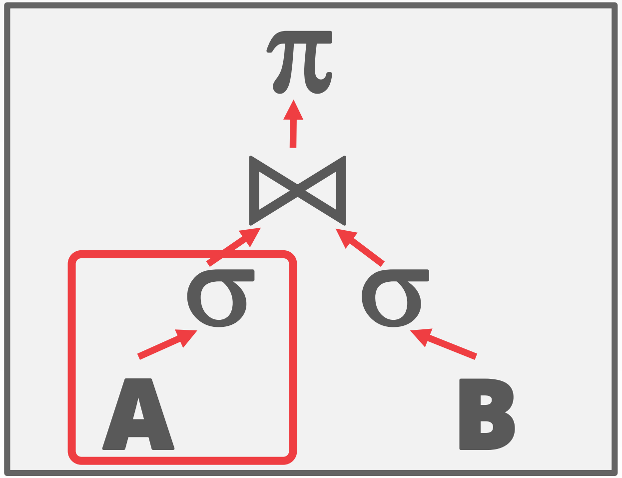
```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```

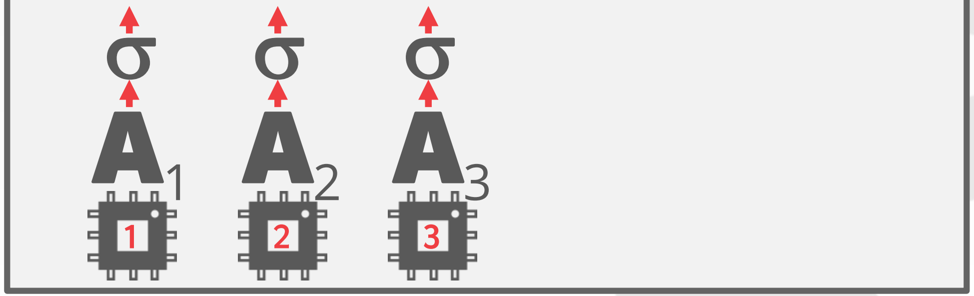




INTRA-OPERATOR PARALLELISM

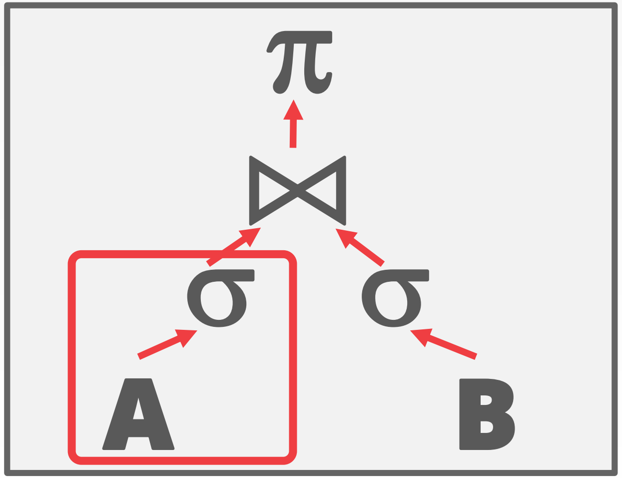
```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```

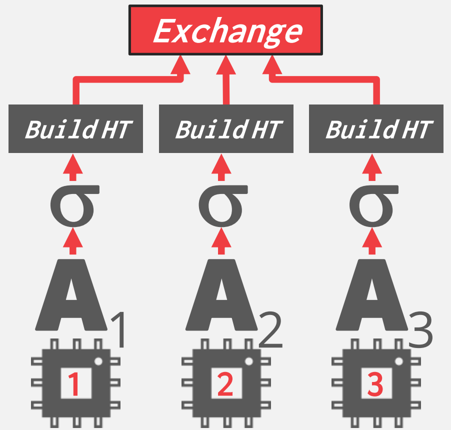




INTRA-OPERATOR PARALLELISM

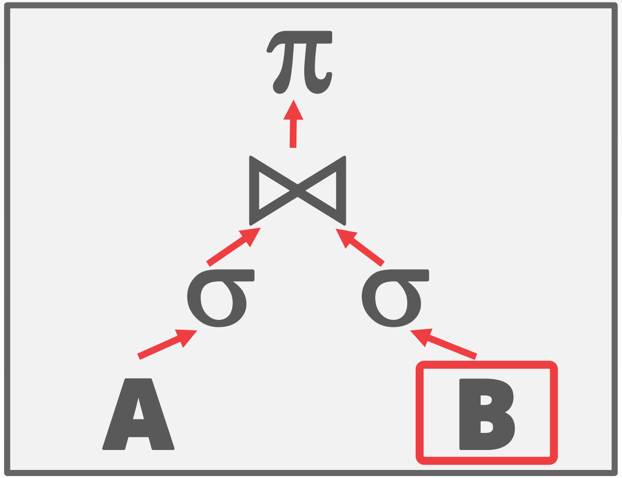
```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```

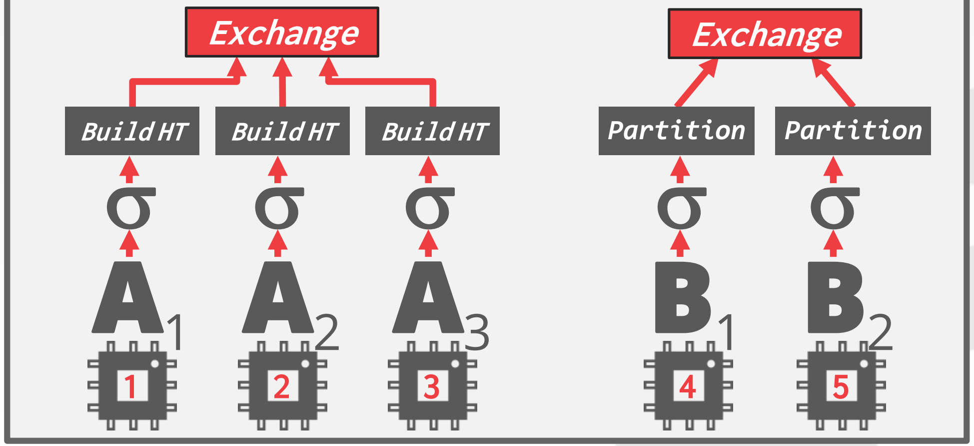




INTRA-OPERATOR PARALLELISM

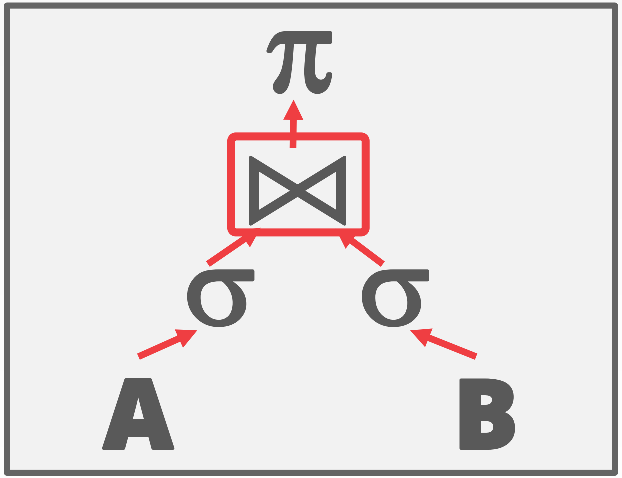
```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```

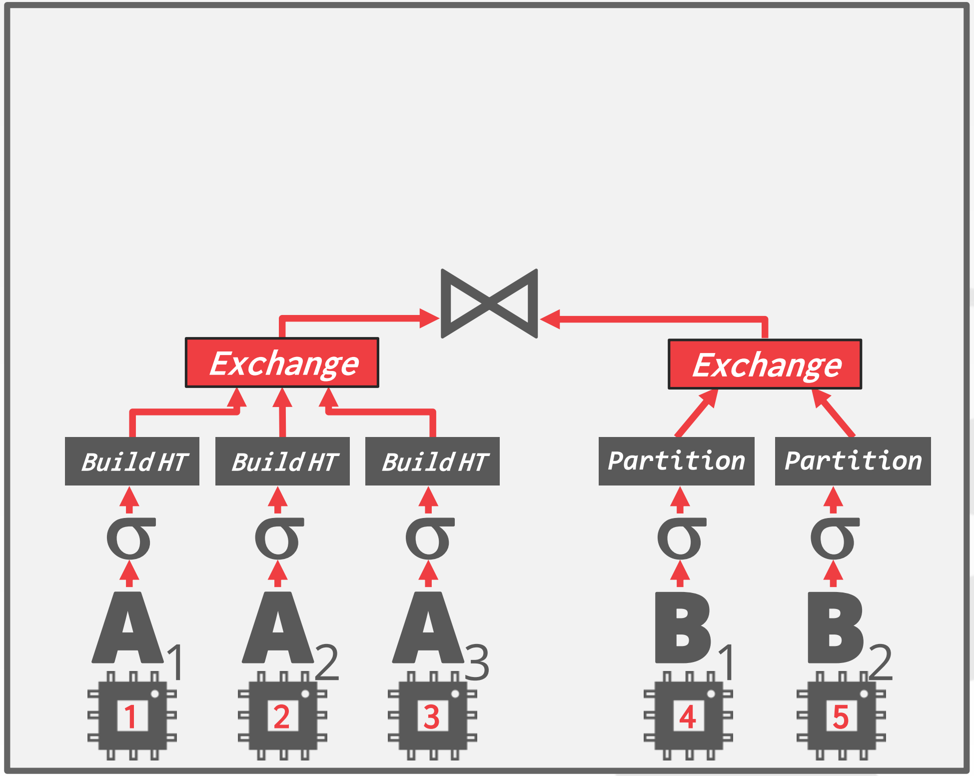




INTRA-OPERATOR PARALLELISM

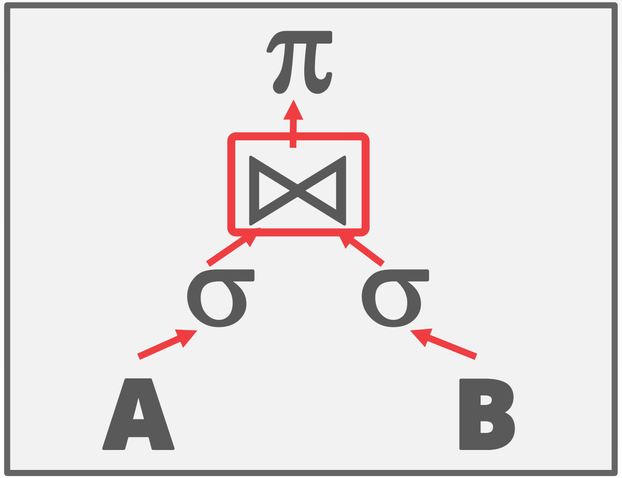
```sql
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```

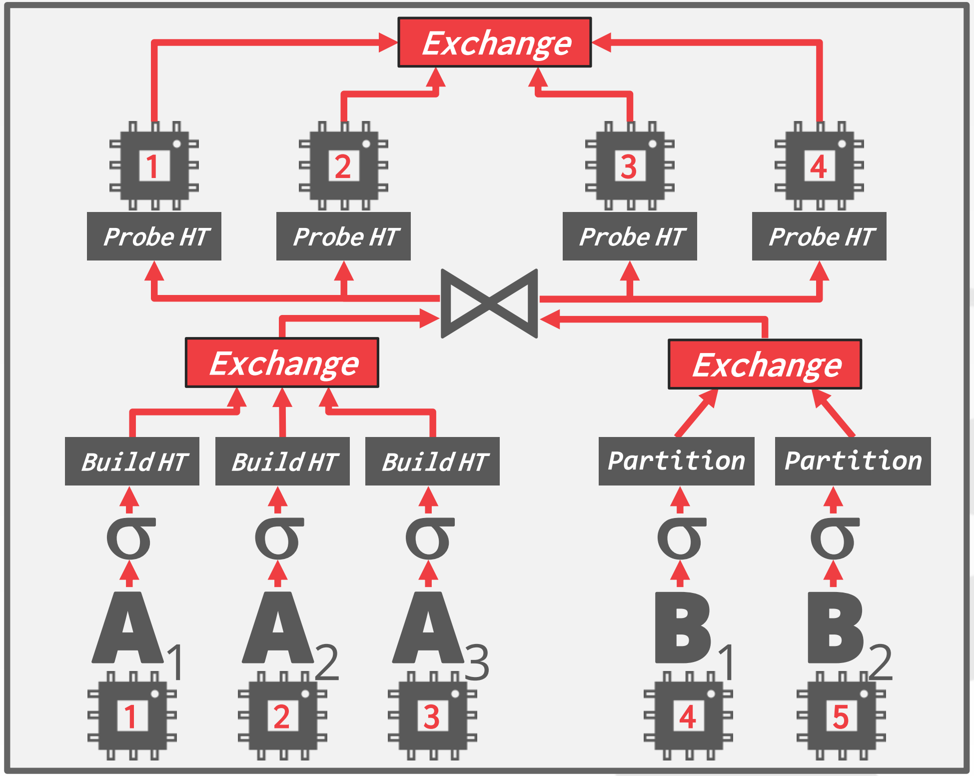




INTRA-OPERATOR PARALLELISM

```sql
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```





INTER-OPERATOR PARALLELISM

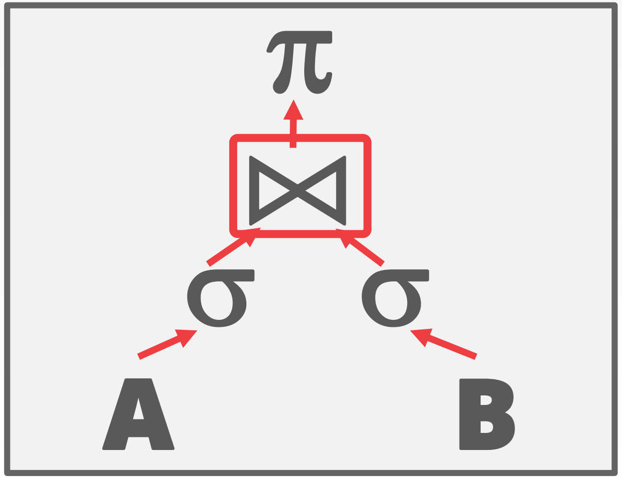
Approach #2: Inter-Operator (Vertical)

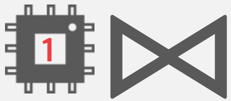
→ Operations are overlapped in order to pipeline data from one stage to the next without materialization.

Also called **pipelined parallelism**.

INTER-OPERATOR PARALLELISM

```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```

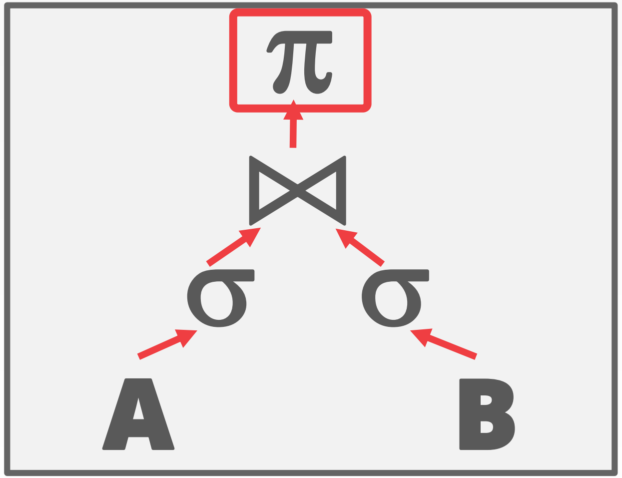


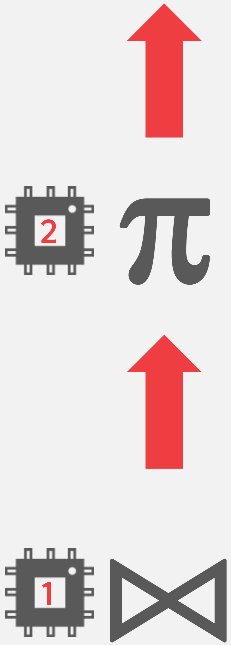



```
for r1 ∈ outer:
  for r2 ∈ inner:
    emit(r1⋈r2)
```

INTER-OPERATOR PARALLELISM

```sql
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```



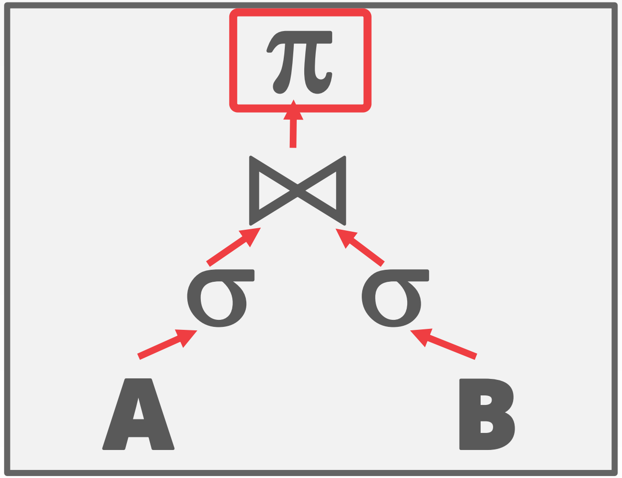


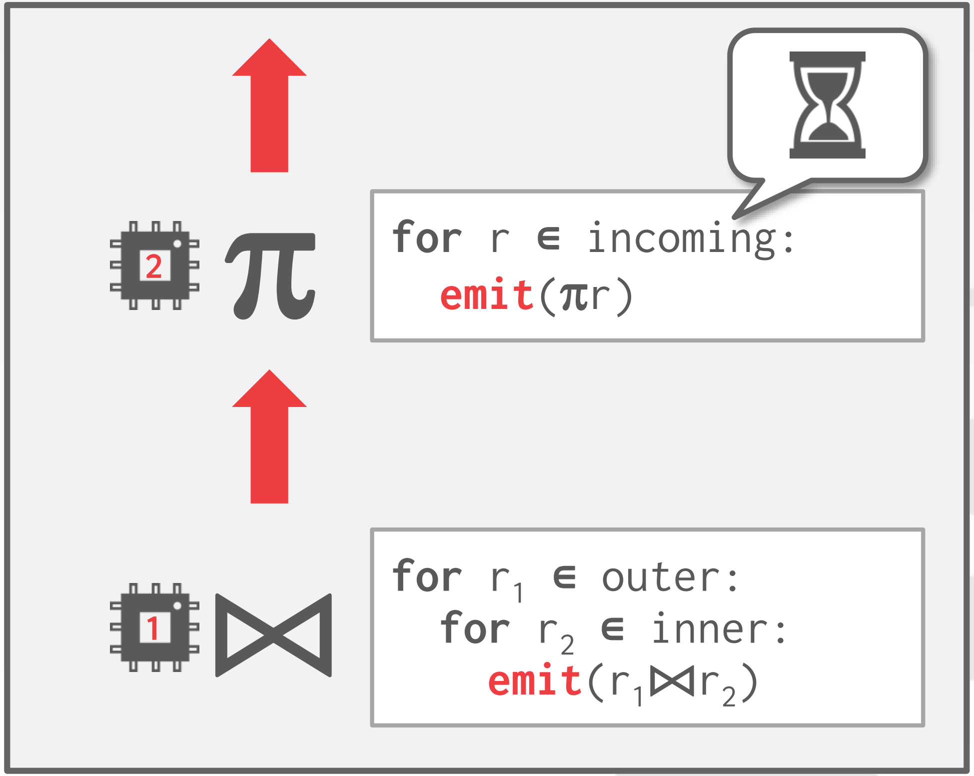

```
for r ∈ incoming:
  emit(πr)
```

```
for r1 ∈ outer:
  for r2 ∈ inner:
    emit(r1⋈r2)
```

INTER-OPERATOR PARALLELISM

```sql
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```





INTER-OPERATOR PARALLELISM

AFAIK, this approach is not widely used in traditional relational DBMSs.

→ Not all operators can emit output until they have seen all of the tuples from their children.

This is more common in <u>**stream processing systems**</u>.

OBSERVATION

Using additional processes/threads to execute queries in parallel won't help if the disk is always the main bottleneck.

→ Can actually make things worse if each worker is reading different segments of disk.

I/O PARALLELISM

Split the DBMS installation across multiple storage devices.

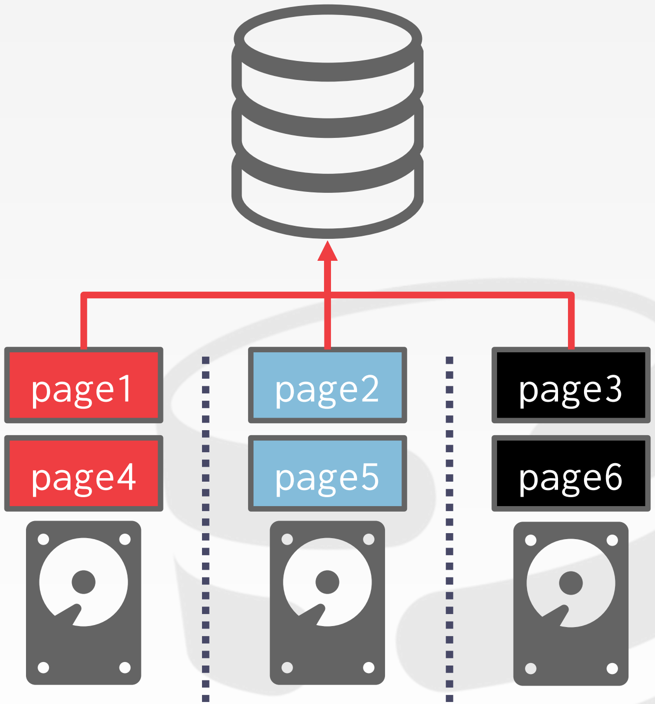
→ Multiple Disks per Database
→ One Database per Disk
→ One Relation per Disk
→ Split Relation across Multiple Disks

MULTI-DISK PARALLELISM

Configure OS/hardware to store the DBMS's files across multiple storage devices.

→ Storage Appliances
→ RAID Configuration

This is transparent to the DBMS.



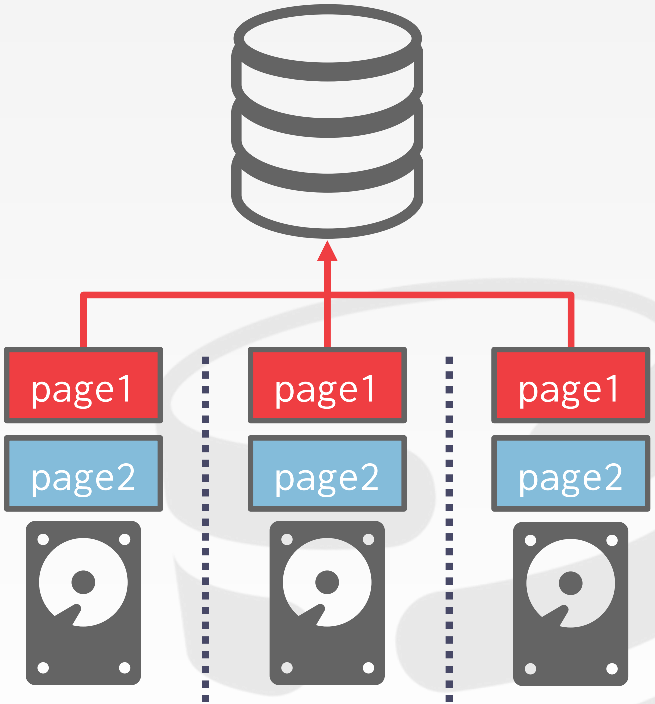
RAID 0 (Stripping)

MULTI-DISK PARALLELISM

Configure OS/hardware to store the DBMS's files across multiple storage devices.

→ Storage Appliances
→ RAID Configuration

This is transparent to the DBMS.



RAID 1 (Mirroring)

DATABASE PARTITIONING

Some DBMSs allow you specify the disk location of each individual database.

→ The buffer pool manager maps a page to a disk location.

This is also easy to do at the filesystem level if the DBMS stores each database in a separate directory.

→ The log file might be shared though

PARTITIONING

Split single logical table into disjoint physical segments that are stored/managed separately.

Ideally partitioning is transparent to the application.

→ The application accesses logical tables and doesn't care how things are stored.
→ Not always true.

VERTICAL PARTITIONING

Store a table's attributes in a separate location (e.g., file, disk volume).

Have to store tuple information to reconstruct the original record.

```
CREATE TABLE foo (
  attr1 INT,
  attr2 INT,
  attr3 INT,
  attr4 TEXT
);
```

Tuple#1	attr1	attr2	attr3	attr4
Tuple#2	attr1	attr2	attr3	attr4
Tuple#3	attr1	attr2	attr3	attr4
Tuple#4	attr1	attr2	attr3	attr4

VERTICAL PARTITIONING

Store a table's attributes in a separate location (e.g., file, disk volume).

Have to store tuple information to reconstruct the original record.

```
CREATE TABLE foo (
  attr1 INT,
  attr2 INT,
  attr3 INT,
  attr4 TEXT
);
```

Partition #1

Tuple#1	attr1	attr2	attr3
Tuple#2	attr1	attr2	attr3
Tuple#3	attr1	attr2	attr3
Tuple#4	attr1	attr2	attr3

Partition #2

Tuple#1	attr4
Tuple#2	attr4
Tuple#3	attr4
Tuple#4	attr4

HORIZONTAL PARTITIONING

Divide the tuples of a table up into disjoint segments based on some partitioning key.

→ Hash Partitioning
→ Range Partitioning
→ Predicate Partitioning

```
CREATE TABLE foo (
  attr1 INT,
  attr2 INT,
  attr3 INT,
  attr4 TEXT
);
```

Tuple#1	attr1	attr2	attr3	attr4
Tuple#2	attr1	attr2	attr3	attr4
Tuple#3	attr1	attr2	attr3	attr4
Tuple#4	attr1	attr2	attr3	attr4

HORIZONTAL PARTITIONING

Divide the tuples of a table up into disjoint segments based on some partitioning key.

→ Hash Partitioning
→ Range Partitioning
→ Predicate Partitioning

```
CREATE TABLE foo (
  attr1 INT,
  attr2 INT,
  attr3 INT,
  attr4 TEXT
);
```

Partition #1

Tuple#1	attr1	attr2	attr3	attr4
Tuple#2	attr1	attr2	attr3	attr4

Partition #2

Tuple#3	attr1	attr2	attr3	attr4
Tuple#4	attr1	attr2	attr3	attr4

CONCLUSION

Parallel execution is important.

(Almost) every DBMS support this.

This is really hard to get right.

→ Coordination Overhead
→ Scheduling
→ Concurrency Issues
→ Resource Contention

NEXT CLASS

How to embed application logic inside of a DBMS to make things go faster:

→ Stored Procedures
→ User-defined Functions
→ User-defined Types
→ Triggers
→ Views

EXTRA CREDIT

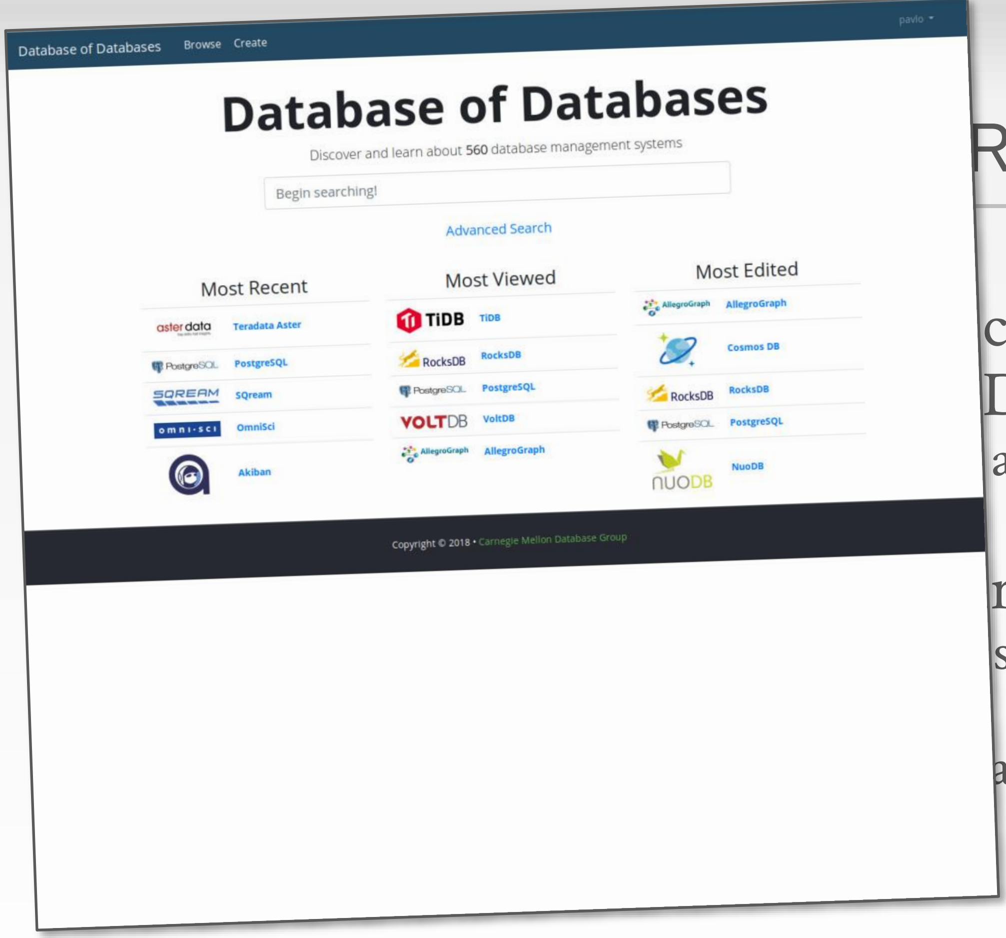
Each student can earn extra credit if they write a encyclopedia article about a DBMS.

→ Can be academic/commercial, active/historical.

Each article will use a standard taxonomy.

→ For each feature category, you select pre-defined options for your DBMS.

→ You will then need to provide a summary paragraph with citations for that category.

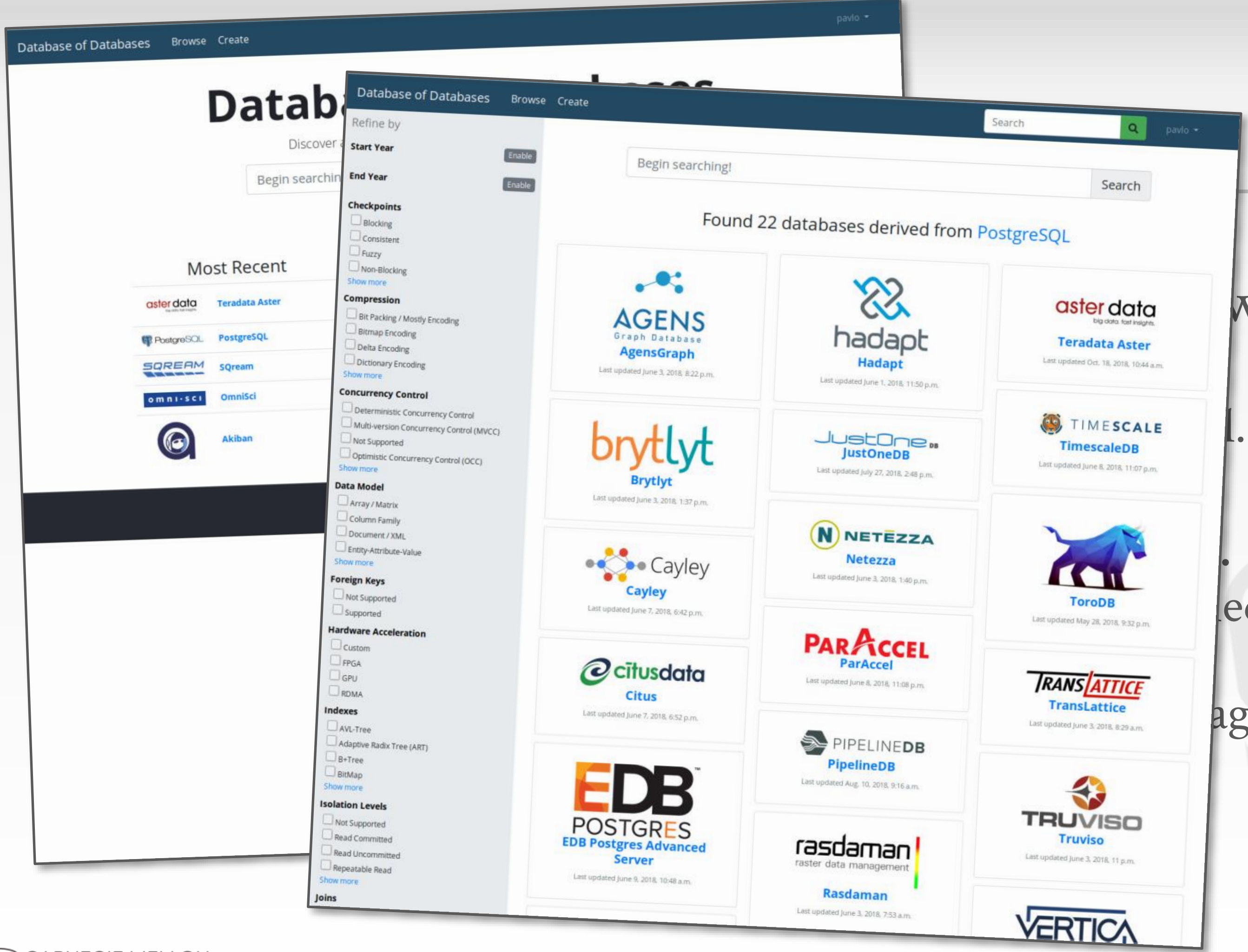


REDIT

credit if they write a
DBMS.
active/historical.

rd taxonomy.
select pre-defined options

a summary paragraph with



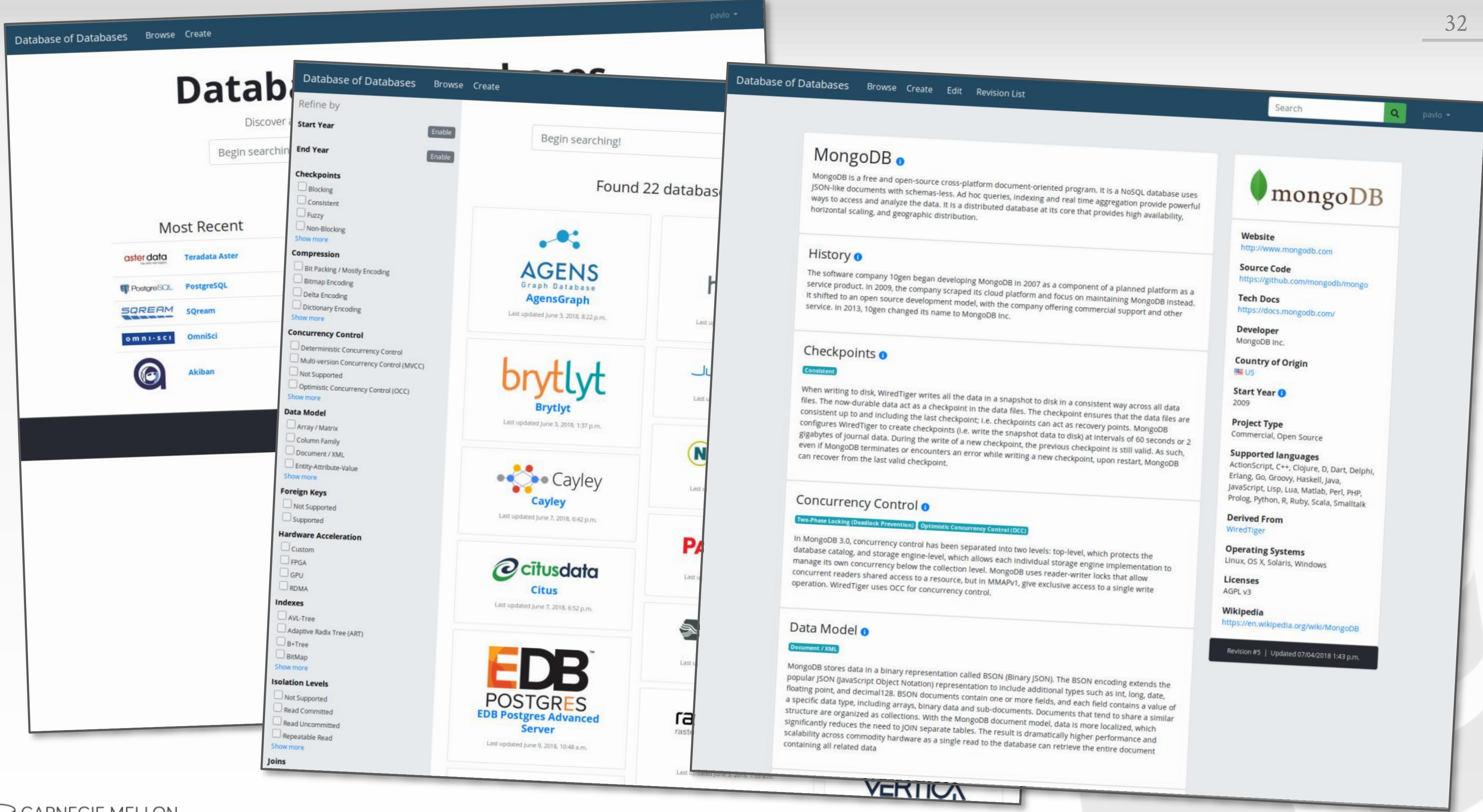
write a

l.

.

ed options

agraph with


Database of Databases
Browse
Create
Discover
Begin searching
Most Recent
Teradata Aster
PostgreSQL
SQream
OmniSci
Akiban
Refine by
Start Year
Enable
End Year
Enable
Checkpoints
Blocking
Consistent
Fuzzy
Non-Blocking
Show more
Compression
Bit Packing / Mostly Encoding
Bitmap Encoding
Delta Encoding
Dictionary Encoding
Show more
Concurrency Control
Deterministic Concurrency Control
Multi-version Concurrency Control (MVCC)
Not Supported
Optimistic Concurrency Control (OCC)
Show more
Data Model
Array / Matrix
Column Family
Document / XML
Entity-Attribute-Value
Show more
Foreign Keys
Not Supported
Supported
Hardware Acceleration
Custom
FPGA
GPU
RDMA
Indexes
AVL-Tree
Adaptive Radix Tree (ART)
B+Tree
BitMap
Show more
Isolation Levels
Not Supported
Read Committed
Read Uncommitted
Repeatable Read
Show more
Joins
Begin searching!
Found 22 databas
AGENS Graph Database
AgensGraph
Last updated June 3, 2018, 8:22 p.m.
brytlyt
Brytlyt
Last updated June 3, 2018, 1:37 p.m.
Cayley
Cayley
Last updated June 7, 2018, 6:42 p.m.
citusdata
Citus
Last updated June 7, 2018, 6:52 p.m.
EDB POSTGRES
EDB Postgres Advanced Server
Last updated June 9, 2018, 10:48 a.m.
Edit
Revision List
Search
pavlo
MongoDB
MongoDB is a free and open-source cross-platform document-oriented program. It is a NoSQL database uses JSON-like documents with schemas-less. Ad hoc queries, indexing and real time aggregation provide powerful ways to access and analyze the data. It is a distributed database at its core that provides high availability, horizontal scaling, and geographic distribution.
History
The software company 10gen began developing MongoDB in 2007 as a component of a planned platform as a service product. In 2009, the company scraped its cloud platform and focus on maintaining MongoDB instead. It shifted to an open source development model, with the company offering commercial support and other service. In 2013, 10gen changed its name to MongoDB Inc.
Checkpoints
Consistent
When writing to disk, WiredTiger writes all the data in a snapshot to disk in a consistent way across all data files. The now-durable data act as a checkpoint in the data files. The checkpoint ensures that the data files are consistent up to and including the last checkpoint; i.e. checkpoints can act as recovery points. MongoDB configures WiredTiger to create checkpoints (i.e. write the snapshot data to disk) at intervals of 60 seconds or 2 gigabytes of journal data. During the write of a new checkpoint, the previous checkpoint is still valid. As such, even if MongoDB terminates or encounters an error while writing a new checkpoint, upon restart, MongoDB can recover from the last valid checkpoint.
Concurrency Control
Two-Phase Locking (Deadlock Prevention)
Optimistic Concurrency Control (OCC)
In MongoDB 3.0, concurrency control has been separated into two levels: top-level, which protects the database catalog, and storage engine-level, which allows each individual storage engine implementation to manage its own concurrency below the collection level. MongoDB uses reader-writer locks that allow concurrent readers shared access to a resource, but in MMAPv1, give exclusive access to a single write operation. WiredTiger uses OCC for concurrency control.
Data Model
Document / XML
MongoDB stores data in a binary representation called BSON (Binary JSON). The BSON encoding extends the popular JSON (JavaScript Object Notation) representation to include additional types such as int, long, date, floating point, and decimal128. BSON documents contain one or more fields, and each field contains a value of a specific data type, including arrays, binary data and sub-documents. Documents that tend to share a similar structure are organized as collections. With the MongoDB document model, data is more localized, which significantly reduces the need to JOIN separate tables. The result is dramatically higher performance and scalability across commodity hardware as a single read to the database can retrieve the entire document containing all related data
mongoDB
Website
http://www.mongodb.com
Source Code
https://github.com/mongodb/mongo
Tech Docs
https://docs.mongodb.com/
Developer
MongoDB Inc.
Country of Origin
US
Start Year
2009
Project Type
Commercial, Open Source
Supported languages
ActionScript, C++, Clojure, D, Dart, Delphi, Erlang, Go, Groovy, Haskell, Java, JavaScript, Lisp, Lua, Matlab, Perl, PHP, Prolog, Python, R, Ruby, Scala, Smalltalk
Derived From
WiredTiger
Operating Systems
Linux, OS X, Solaris, Windows
Licenses
AGPL v3
Wikipedia
https://en.wikipedia.org/wiki/MongoDB
Revision #5 | Updated 07/04/2018 1:43 p.m.
VERTICA

DBDB.IO

All the articles will be hosted on our new website.
→ I will post the user/pass on Piazza.

I will post a sign-up sheet for you to pick what DBMS you want to write about.
→ If you choose a widely known DBMS, then the article will need to be comprehensive.
→ If you choose an obscure DBMS, then you will have do the best you can to find information.

☠ PLAGIARISM WARNING ☠

This article must be your own writing with your own images. You may **not** copy text/images directly from papers or other sources that you find on the web.

Plagiarism will **not** be tolerated.
See CMU's Policy on Academic Integrity for additional information.